# An Efficient Approximation for the Generalized Assignment Problem

Reuven Cohen Liran Katzir Danny Raz
Department of Computer Science
Technion
Haifa 32000, Israel

**Abstract**

We present a simple family of algorithms for solving the Generalized Assignment Problem (GAP). Our technique is based on a novel combinatorial translation of any algorithm for the knapsack problem into an approximation algorithm for GAP. If the approximation ratio of the knapsack algorithm is $\alpha$ and its running time is $O(f(N))$, our algorithm guarantees a $(1+\alpha)$ approximation ratio, and it runs in $O(M \cdot f(N) + M \cdot N)$, where $N$ is the number of items and $M$ is the number of bins. Not only does our technique comprise a general interesting framework for the GAP problem; it also matches the best combinatorial approximation for this problem, with a much simpler algorithm and a better running time.

**Keywords:** Generalized Assignment Problem, local ratio, approximation algorithms.

## 1 Introduction

We study the following maximization version of the Generalized Assignment Problem:

**Instance:** A pair $(B, S)$ where $B$ is a set of $M$ bins (knapsacks) and $S$ is a set of $N$ items. Each bin $C_j \in B$ has capacity $c(j)$, and for each item $i$ and bin $C_j$ we are given a size $s(i,j)$ and a profit $p(i,j)$.

**Objective:** Find a subset $U \subseteq S$ of items that has a feasible packing in $B$, such that the profit is maximized.

GAP is applicable in many fields, including data storage and retrieval in disks [14], inventory matching [8], and distributed caching [9]. The first known approximation algorithm for GAP is an LP-based 2-approximation algorithm, presented implicitly in [13]. Later, this algorithm was explicitly presented in [7], which also presents APX-hardness proofs for two special cases. Moreover, the authors prove that a greedy algorithm that iteratively fills each bin using an FPTAS for the single knapsack has a constant approximation ratio for two special cases: (a) $\frac{e}{e-1}+\epsilon$ for every $\epsilon$, when all bins have the same size and each item has the same profit for all bins; (b) $2+\epsilon$ for every $\epsilon$, when bins may vary in size and each item has the same profit for all bins. In [9] the authors showed that two extensions of GAP cannot be approximated within $(\frac{e}{e-1})$ unless $NP \subseteq DTIME(n^{O(\log\log n)})$.

Several papers address special cases of GAP [8, 14, 12]. Ref. [3] provides a family of algorithms for solving resource allocation problems. Although not specified explicitly in [3], their approach will yield a $(2+\epsilon)$-approximation algorithm that runs in $O((N \cdot M)^2/\epsilon)$. A combinatorial local search algorithm with a $(2+\epsilon)$-approximation guarantee and an LP-based algorithm with an $(\frac{e}{e-1}+\epsilon)$-approximation guarantee are presented in [9]. The $(\frac{e}{e-1}+\epsilon)$-approximation is the best known approximation for GAP. The combinatorial local search algorithm of [9] uses an algorithm $A$ for the single knapsack problem. The running time of the local search algorithm is $O(M \cdot f(N) \cdot \frac{N \ln(1/\epsilon)}{\beta^{-1}+1})$, where $\beta$ is the approximation ratio of $A$.

In this short paper we present a combinatorial algorithm for GAP that has the same approximation ratio as the combinatorial local search algorithm presented in [9], but with a better running time. Our algorithm is actually a family of algorithms produced by applying the local-ratio technique [4] to any algorithm for the single bin (knapsack) problem. Any $\alpha$-approximation algorithm $A$ for the knapsack problem can be transformed into a $(1+\alpha)$-approximation algorithm for GAP. The running time of our algorithm is $O(M \cdot f(N) + M \cdot N)$, where $f(N)$ is the running time of algorithm $A$. Specifically, the greedy 2-approximation for knapsack, which runs in $O(N \log N)$ time using sorting or $O(N)$ using weighted linear selection, is translated into a 3-approximation algorithm whose running time is $O(NM \log N)$ or $O(NM)$ respectively. The FPTAS for knapsack described in [11] will be translated into a $(2+\epsilon)$-approximation algorithm that runs in $O(MN \log \frac{1}{\epsilon} + \frac{M}{\epsilon^4})$.

The rest of this paper is organized as follows. In section 2 we present the new algorithm. In section 3 we discuss its possible implementations and in section 4 we conclude the paper.

## 2 The new algorithm

We shall use the "local-ratio technique" as proposed in [4] and extended in [1, 3, 5]. For the sake of completeness, this technique is outlined here.

Let $w(x)$ be a profit function and let $F$ be a set of feasibility constraints on a solution $x$. Solution $x$ is a *feasible solution* to a given problem $(F, w())$ if it satisfies all the constraints in $F$. The *value* of a feasible solution $x$ is $w(x)$. A feasible solution is *optimal* for a maximization problem if its value is maximal among all feasible solutions. A feasible solution $x$ is an *$r$-approximate* solution if $r \cdot w(x) \geq w(x^*)$, where $x^*$ is an optimal solution. An algorithm is said to be an $r$-approximation algorithm if it always computes $r$-approximate solutions.

**Theorem 2.1 (Local Ratio).** *Let $F$ be a set of constraints and let $w()$, $w_1()$, and $w_2()$ be profit functions such that $w() = w_1() + w_2()$. Then, if $x$ is an $r$-approximate solution with respect to $(F, w_1())$ and with respect to $(F, w_2())$, it is also an $r$-approximate solution with respect to $(F, w())$.*

*Proof:* [4] Let $x^*$, $x_1^*$ and $x_2^*$ be optimal solutions for $(F, w())$, $(F, w_1())$, and $(F, w_2())$ respectively. Then $w(x) = w_1(x) + w_2(x) \geq r \cdot w_1(x_1^*) + r \cdot w_2(x_2^*) \geq r \cdot (w_1(x^*) + w_2(x^*)) = r \cdot w(x^*)$. ■

**Algorithm 1.** *Let $M$ be the number of bins and $N$ be the number of items. Let $p$ be an $N \times M$ profit matrix. The value of $p[i, j]$ indicates the profit of item $i$ when selected for bin $C_j$. Let $A$ be an algorithm for the knapsack problem. We now construct from $A$ an algorithm for GAP. Since our algorithm modifies the profit function, we use the notation $p_j$ to indicate the profit matrix at the $j$th recursive call. Initially, we set $p_1 \leftarrow p$, and we invoke the following Next-Bin procedure with j=1:*

*Procedure Next-Bin(j)*

1. *Run Algorithm $A$ on bin $C_j$ using $p_j$ as the profit function, and let $\bar{S}_j$ be the set of selected items returned.*

2. *Decompose the profit function $p_j$ into two profit functions $p_j^1$ and $p_j^2$ such that for every $k$ and $i$, where $1 \leq k \leq M$ and $1 \leq i \leq N$,*

$$p_j^1[i,k] = \begin{cases} p_j[i,j] & \textit{if } (i \in \bar{S}_j) \textit{ or } (k = j) \\ 0 & \textit{Otherwise} \end{cases} \quad \text{and} \quad p_j^2 = p_j - p_j^1.$$

   *This implies that $p_j^1$ is identical to $p_j$ with regard to bin $C_j$; in addition, if item $i \in \bar{S}_j$, then $i$ is assigned in $p_j^1$ the same profit $p_j[i,j]$ for all the bins. All the other entries are zeros.*

3. *If $j < M$ then*

   - *set $p_{j+1} \leftarrow p_j^2$, and remove the column of bin $C_j$ from $p_{j+1}$.*
   - *Perform Next-Bin(j+1). Let $S_{j+1}$ be the returned assignment list.*
   - *Let $S_j$ be the same as $S_{j+1}$ except that it also assigns to bin $C_j$ all the items in $\bar{S}_j \setminus \bigcup_{i=j+1}^{M} S_i$.*

   - *Return $S_j$.*

   *else, return $S_j = \bar{S}_j$.* ■

Figure 1 illustrates a running example of the algorithm on 4 items $\{i_1, i_2, i_3, i_4\}$ and 3 bins $\{C_1, C_2, C_3\}$. To simplify the example, at each step algorithm $A$ chooses the optimal solution with respect to $p$.

The algorithm clearly stops. The returned solution must be a valid assignment since no item can be assigned to two or more different bins, and if $\bar{S}_j$ fits the size of bin $C_j$, so must any subset of $\bar{S}_j$.

**Claim 2.1.** *If Algorithm $A$ is a $\alpha$-approximation for the knapsack problem, then Algorithm 1 is a $(1+\alpha)$-approximation for GAP.*

*Proof:* For the following proof we use the notation $p(S)$ to indicate the profit gained by assignment $S$. The proof is by induction on the number of bins available when the algorithm is invoked. When there is a single bin, the assignment returned by the algorithm, $\bar{S}_M$, is a $(\alpha)$-approximation due to Algorithm $A$'s validity, and therefore a $(1+\alpha)$-approximation with respect to $p_M$. For the inductive step, assume that $S_{j+1}$ is a $(1+\alpha)$-approximation with respect to $p_{j+1}$, and we shall now prove that $S_j$ is also a $(1+\alpha)$-approximation with respect to $p_j^2$. Matrix $p_j^2$ is identical to $p_{j+1}$, except that it contains a column with profit 0. $S_{j+1}$ is also an $(1+\alpha)$-approximation with respect to $p_j^2$, as is $S_j$ because it contains the items assigned by $S_{j+1}$.

Profit matrix $p_j^1$ has three components: (1) a column for bin $C_j$, which is identical to $C_j$'s column in $p_j$, (2) the rows of items selected for $\bar{S}_j$, whose profit in all the columns is identical to their profit in the column of $C_j$, and (3) the remaining slots, all of which contain zeros. We shall now show an upper bound on the profit of any assignment with respect to $p_j^1$. Only components (1) and (2) of $p_j^1$ can contribute profit to an assignment. By the validity of algorithm $A$, $\bar{S}_j$ is an $\alpha$-approximation for bin $C_j$ with respect to component (1). Therefore the best possible solution with respect to component (1) will gain at most $\alpha \cdot p_j^1(\bar{S}_j)$. The best possible solution with respect

$$\underline{\text{item sizes}} \qquad\qquad\qquad\qquad \underline{\text{profit function}}$$

$$s = \begin{array}{c|c|c|c|} & C_1 & C_2 & C_3 \\ \hline i_1 & 1 & 1 & 1 \\ \hline i_2 & 2 & 3 & 3 \\ \hline i_3 & 2 & 3 & 4 \\ \hline i_4 & 1 & 2 & 3 \\ \hline \end{array} \qquad \underline{\text{bin sizes}} \quad \text{c} = \begin{array}{ccc} C_1 & C_2 & C_3 \\ (2\ , & 3\ , & 4) \end{array} \qquad p = \begin{array}{c|c|c|c|} & C_1 & C_2 & C_3 \\ \hline i_1 & 3 & 1 & 5 \\ \hline i_2 & 1 & 1 & 1 \\ \hline i_3 & 5 & 15 & 25 \\ \hline i_4 & 25 & 15 & 5 \\ \hline \end{array}$$

$$p_1 = \begin{array}{c|c|c|c|} & C_1 & C_2 & C_3 \\ \hline i_1 & 3 & 1 & 5 \\ \hline i_2 & 1 & 1 & 1 \\ \hline i_3 & 5 & 15 & 25 \\ \hline i_4 & 25 & 15 & 5 \\ \hline \end{array} \qquad \bar{S}_1 = \{i_1, i_4\} \qquad p_1^1 = \begin{array}{c|c|c|c|} & C_1 & C_2 & C_3 \\ \hline i_1 & 3 & 3 & 3 \\ \hline i_2 & 1 & 0 & 0 \\ \hline i_3 & 5 & 0 & 0 \\ \hline i_4 & 25 & 25 & 25 \\ \hline \end{array} \qquad p_1^2 = \begin{array}{c|c|c|c|} & C_1 & C_2 & C_3 \\ \hline i_1 & 0 & -2 & 2 \\ \hline i_2 & 0 & 1 & 1 \\ \hline i_3 & 0 & 15 & 25 \\ \hline i_4 & 0 & -10 & -20 \\ \hline \end{array}$$

$$p_2 = \begin{array}{c|c|c|} & C_2 & C_3 \\ \hline i_1 & -2 & 2 \\ \hline i_2 & 1 & 1 \\ \hline i_3 & 15 & 25 \\ \hline i_4 & -10 & -20 \\ \hline \end{array} \qquad \bar{S}_2 = \{i_3\} \qquad p_2^1 = \begin{array}{c|c|c|} & C_2 & C_3 \\ \hline i_1 & -2 & 0 \\ \hline i_2 & 1 & 0 \\ \hline i_3 & 15 & 15 \\ \hline i_4 & -10 & 0 \\ \hline \end{array} \qquad p_2^2 = \begin{array}{c|c|c|} & C_2 & C_3 \\ \hline i_1 & 0 & 2 \\ \hline i_2 & 0 & 1 \\ \hline i_3 & 0 & 10 \\ \hline i_4 & 0 & -20 \\ \hline \end{array}$$

$$p_3 = \begin{array}{c|c|} & C_3 \\ \hline i_1 & 2 \\ \hline i_2 & 1 \\ \hline i_3 & 10 \\ \hline i_4 & -20 \\ \hline \end{array} \qquad \bar{S}_3 = \{i_3\} \qquad p_3^1 = \begin{array}{c|c|} & C_3 \\ \hline i_1 & 2 \\ \hline i_2 & 1 \\ \hline i_3 & 10 \\ \hline i_4 & -20 \\ \hline \end{array} \qquad p_3^2 = \begin{array}{c|c|} & C_3 \\ \hline i_1 & 0 \\ \hline i_2 & 0 \\ \hline i_3 & 0 \\ \hline i_4 & 0 \\ \hline \end{array}$$

$S_3 = (\{i_3\})$, $S_2 = (\{\}, \{i_3\})$, $S_1 = (\{i_1, i_4\}, \{\}, \{i_3\})$, final assignment:= $(\{i_1, i_4\}, \{\}, \{i_3\})$

Figure 1: Algorithm 1 – running example for 4 items

to component (2) will gain at most $p_j^1(\bar{S}_j)$, since the profit in $p_j^1$ of items selected by $\bar{S}_j$ is the same regardless of where they are assigned. This implies that $\bar{S}_j$ is a $(1+\alpha)$-approximation with respect to $p_j^1$. According to the last step of the algorithm, it is clear that the items selected for $\bar{S}_j$ are a subset of items selected by $S_j$. Therefore $p_j^1(S_j) \geq p_j^1(\bar{S}_j)$ (they are actually equal), and $S_j$ is a $(1+\alpha)$-approximation with respect to $p_j^1$. Since $S_j$ is a $(1+\alpha)$-approximation with respect to both $p_j^1$ and $p_j^2$, and since $p_j = p_j^1 + p_j^2$, by Theorem 2.1, it is also a $(1+\alpha)$-approximation with respect to $p_j$. ■

The algorithm approximation ratio is tight. For example, consider the case of two 1-unit bins and two items. Let the first item have profit $\alpha > 1$ in the first bin and 0 in the second. Let the second item have profit 1 in both bins. The algorithm might return the solution with the second item in the first bin, where the optimal solution is to have the first item in the first bin and the second item in the second bin.

## 3 Implementation Notes

To discuss the running time of the algorithm, we present in this section an iterative implementation. For this implementation we use the vector $P_j$, $j = 1 \dots M$, to denote the profit function for bin $C_j$. In addition, we use a vector $T$ to indicate the status of each item: $T[i] = -1$ indicates that item $i$ has not been selected yet, whereas $T[i] = j$ indicates that this item has been selected for bin $C_j$, and $C_j$ is the last bin for which it was selected. We later show that this iterative version is equivalent to Algorithm 1.

**Algorithm 2 (An iterative version of Algorithm 1).**

```
1. Initialization: $\forall i = 1 \dots N$, set $T[i] \leftarrow -1$.
2. For $j = 1 \dots M$ do:
   (a) $P_j$ creation: For $i = 1 \dots N$, set $P_j[i] \leftarrow \begin{cases} p[i,j] & \text{if } T[i] = -1; \\ p[i,j] - p[i,k] & \text{if } T[i] = k. \end{cases}$
   (b) Run Algorithm $A$ on bin $C_j$ using $P_j$. Let $\bar{S}_j$ be the set of selected items returned.
   (c) $\forall i \in \bar{S}_j$, set $T[i] \leftarrow j$.
3. In the returned assignment, if $T[i] \neq -1$ then item $i$ is mapped to bin $T[i]$.
```

We now show that Algorithm 2 and Algorithm 1 are identical, using the following three steps:

(1) Both algorithms use the same profit function as input to Algorithm $A$. This is proven in Lemma 3.1.

(2) Given that both algorithms use the same profit function, the value of $\bar{S}_j$ computed by Algorithm 1 is identical to the value of $\bar{S}_j$ computed by Algorithm 2. This follows from the fact that the same Algorithm $A$ is used as a procedure by both algorithms.

(3) Given that the value of $\bar{S}_j$ computed by Algorithm 1 is identical to the value of $\bar{S}_j$ computed by Algorithm 2, the final assignment returned by both algorithms is identical. The reason is that for both algorithms the merge of $\bar{S}_j$, $\forall j = 1 \dots N$, is performed in descending order. If item $i$ has been selected by Algorithm $A$ for multiple bins, it will finally be assigned to the bin with the highest index.

**Lemma 3.1.** *At each iteration $j$, the following holds: $\forall i = 1 \dots N$, $P_j[i] = p_j[i,j]$.*

*Proof:* $P_j[i]$ is calculated for two cases: (1) $T[i] = -1$, that is, item $i$ has not been selected for bins $C_1, C_2, \dots, C_{j-1}$; and (2) $T[i] = k$, that is, item $i$ has been selected at least once, and bin $C_k$ is the last bin among $C_1, C_2, \dots, C_{j-1}$ to which item $i$ was assigned.

For the first case, since $P_j[i]$ is set as $p[i,j]$, we need to prove that $p_j[i,j] = p[i,j]$. Since item $i$ is not selected for any $\bar{S}_k$ where $k < j$, then $p_k^1[i,j] = 0$. Hence $p_{k+1}[i,j] = p_k[i,j]$. Since $p_1[i,j] = p[i,j]$, we get $p_j[i,j] = p_{j-1}[i,j] = \dots = p_1[i,j] = p[i,j]$.

For the second case, since $P_j[i]$ is set as $p[i,j] - p[i,k]$, we need to prove that for Algorithm 1 the following holds:

$$p_j[i,j] = p[i,j] - p[i,k], \tag{1}$$

where item $i$ was selected for the last time before the $j$th iteration for bin $C_k$, where $k \leq j-1$.

Observe that for $j+1 \leq q$ the following holds:

$$p_{j+1}[i,q] = p_j^2[i,q] = p_j[i,q] - p_j^1[i,q].$$

The first equality holds since $p_j^2$ is identical to $p_{j+1}$, except that it holds a column for bin $C_j$, but $j+1 \leq q$. The second equality follows directly from the algorithm. We apply this equation iteratively until $j=1$ and get that for $j+1 \leq q$ the following holds:

$$p_{j+1}[i,q] = p_j[i,q] - p_j^1[i,q] = p_{j-1}[i,q] - p_j^1[i,q] - p_{j-1}^1[i,q] = \ldots = p_1[i,q] - \sum_{h=1}^{j} p_h^1[i,q].$$

Since $p[i,j] = p_1[i,j]$ holds for every $i$ and $j$, we get $p_{j+1}[i,q] = p[i,q] - \sum_{h=1}^{j} p_h^1[i,q]$ for $j+1 \leq q$.

Now, let $L_j = \{l_1, l_2, \ldots l_p\}$ be the set of indices of iterations during which item $i$ is selected until (including) the $j$th iteration. We know that $p_h^1[i,q] = 0$ holds for every $h \notin L_j$, and that $p_h^1[i,q] = p_h[i,h]$ holds for every $h \in L_j$. Therefore, for every $j+1 \leq q$ we get:

$$p_{j+1}[i,q] = p[i,q] - \sum_{h \in L_j} p_h^1[i,q] = p[i,q] - \sum_{h \in L_j} p_h[i,h]. \qquad (2)$$

Let $x_j$ be the last index in $L_j$. Note that $L_{x_j-1} = L_j \setminus \{x_j\}$. Hence, Eq.2 can be written as:

$$p_{j+1}[i,q] = p[i,q] - p[i,x_j] + p[i,x_j] - p_{x_j}[i,x_j] - \sum_{h \in L_{x_j-1}} p_h[i,h]. \qquad (3)$$

Since Eq.2 holds for every $j$ and $q$ such that $j \leq q-1$, it also holds for $j = x_j - 1$ and $q = x_j$. Therefore, we get:

$$p_{x_j}[i,x_j] = p[i,x_j] - \sum_{h \in L_{x_j-1}} p_h[i,h]. \qquad (4)$$

Substituting Eq.4 into Eq.3 yields that $p_{j+1}[i,q] = p[i,q] - p[i,x_j]$. Substituting $q = j+1$ yields $p_{j+1}[i,j+1] = p[i,j+1] - p[i,x_j]$. By replacing $j+1$ with $j$ we get $p_j[i,j] = p[i,j] - p[i,x_{j-1}]$, where $x_{j-1}$ is the last iteration (bin) for which item $i$ is selected before (including) the $(j-1)$th iteration, thereby proving Eq.1.

■

The total running time complexity of Algorithm 2 is $O(M \cdot f(N) + M \cdot N)$, where $f(N)$ is the running time of algorithm $A$.

Note that a practical implementation of this algorithm might benefit from reassigning the items in bin $C_j$ when procedure Next-Bin returns from the $(j+1)$th call. A tighter approximation can be calculated as the algorithm returns from the recursive calls by maintaining the profit of the assignment against an upper bound of the optimum.

## 4 Conclusions and Extensions

We presented an efficient algorithm for GAP and showed a simple way to implement it. Our algorithm guarantees an $(1+\alpha)$-approximation solution, and its time complexity is $O(M \cdot f(N) +$

$M \cdot N$), where $N$ is the number of items, $M$ is the number of bins, and $O(f(N))$ is the time complexity of an $\alpha$-approximation algorithm for Knapsack, used as a sub-routine.

Our algorithm requires only that if $\bar{S}_j$ is a feasible solution to bin $C_j$, then so is any subset of $\bar{S}_j$. Therefore, this algorithm is also applicable for other problems, like the "Multiple Choice Multiple Knapsacks" problem [10], where there are multiple knapsacks and each item might have a different weight and profit not only for different knapsacks but also for the same knapsack, or the Separable Assignment Problem (SAP) [9]. Moreover, for a single machine, the algorithm in Ref.[2] can also be used, with or without preemption, to solve the corresponding unrelated machine problem (see [6]). As this algorithm is optimal ($\alpha = 1$), our algorithm guarantees a 2-approximation solution for maximizing the number of satisfied jobs, which is the best known result to the best of our knowledge.

## References

[1] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIJDM: SIAM Journal on Discrete Mathematics*, 12:289–297, 1999.

[2] P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2:245–252, 1999.

[3] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

[4] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–45, 1985.

[5] R. Bar-Yehuda, M. M. Halldórsson, J. S. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. In *SODA'02: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 732–741, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[6] P. Brucker. *Scheduling Algorithms.* Springer, Berlin, 1998.

[7] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *SODA'00: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 213–222, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[8] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi. Approximation algorithms for the multiple knapsack problem with assignment restrictions. *J. Comb. Optim.*, 4(2):171–186, 2000.

[9] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *SODA'06: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 611–620, 2006.

[10] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.

[11] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979.

[12] Z. Nutov, I. Beniaminy, and R. Yuster. A (1-1/e)-approximation algorithm for the generalized assignment problem. *Oper. Res. Lett.*, 34(3):283–288, 2006.

[13] D. B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62(3):461–474, 1993.

[14] F. C. Spieksma, J. Aerts, and J. Korst. An approximation algorithm for a generalized assignment problem with small resource requirements. In *ORBEL: 18th Belgian Conference on Quantitative Methods for Decision Making*, 2004.